Name: Noah Wagner, Dataset: https://www.kaggle.com/datasets/whigmalwhim/steam-releases/

In [53]:
```
import numpy as np
import pandas as pd

from sklearn.pipeline import Pipeline
from sklearn.base import TransformerMixin, BaseEstimator
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import GradientBoostingClassifier
```

In [54]:
```
data = pd.read_csv("game_data_trimmed.csv")
data = data[["release", "peak_players", "total_reviews", "rating", "players_right_now"]]
data.fillna(value=0, inplace = True)

#edit players_right_now column to be numerical (has strings such as "1,234")
data["players_right_now"] = data["players_right_now"].apply(lambda x: int(x.replace(",", "")) if isinstance(x, str) else x)
#modify release dates to be numerical
data["release"] = data["release"].apply(lambda x: int(x.replace("-", "")))

data["is_popular"] = data["players_right_now"] > 10

xs = data[["release", "peak_players", "total_reviews", "rating"]]
ys = data["is_popular"]

print(xs, ys)
```

```
       release  peak_players  total_reviews  rating
0     20230126          4529          20034   96.39
1     20230324        168191          63368   95.75
2     20230331         15543          12856   95.54
3     20230328          1415          11926   95.39
4     20230125          6132          14476   95.09
...        ...           ...            ...     ...
9995  20221104             1              3   67.06
9996  20221111             2              3   67.06
9997  20220905             2              3   67.06
9998  20220804             1              3   67.06
9999  20221107             5              3   67.06

[10000 rows x 4 columns] 0        True
1        True
2        True
3        True
4        True
        ...
9995    False
9996    False
9997    False
9998    False
9999    False
Name: is_popular, Length: 10000, dtype: bool
```

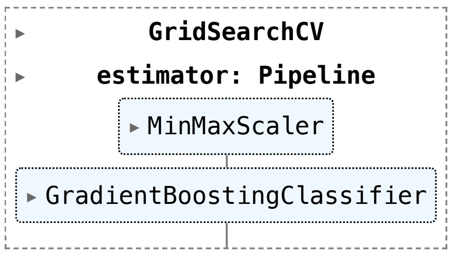
In [55]:
```
steps = [
    ("scale", MinMaxScaler()),
    ("classify", GradientBoostingClassifier())
]
pipe = Pipeline(steps)

grid = {
        "classify__max_depth": [1, 2, 3],
    "classify__max_features": [1, 2, 3, 4],
        "classify__learning_rate": [0.01, 0.025, 0.05],
}

search = GridSearchCV(pipe, grid, scoring = "f1", n_jobs=-1)

search.fit(xs, ys)
```

Out[55]:



▸ GridSearchCV
▸ estimator: Pipeline
▸ MinMaxScaler
▸ GradientBoostingClassifier

In [56]:
```
print(search.best_score_)
print(search.best_params_)
```

```
0.7135327573543893
{'classify__learning_rate': 0.025, 'classify__max_depth': 1, 'classify__max_features': 3}
```

Unlike in test 1, this time we used cross validation. If we went back to the train/test split we used before, would you expect your chosen metric to increase or decrease?

Using a train/test split could lead to a higher variance in our models performance. Without cross validation, we could get slightly lucky or unlucky in how our train / test dataset was selected. So on average, I think my metric would remain roughly the same, but have a higher variance if I used a train / test split.

Why did you choose this metric? Why is it appropriate for your classification task and data?

I chose the F score as my metric because I couldn't simply use accuracy since my data was heavily skewed. The target column has eight times more False instances than True. I also can't simply use precision or recall, since those alone wouldn't give an accurate account of how my model is performing. However, using a metric that combines the two (F score) is exactly what I need. It represents accuracy but isn't affected by a skewed dataset.

Why do you think the hyperparameters that were selected by the grid were optimal? Were any of the results surprising? Why or why not?

On average, the parameters my model settles on are a max_depth of 1, max_features of 3, and a learning_rate of 0.025. I'm not very suprized on how simple the trees are (after all, the idea of gradient boosting is to have simple trees in succession), since the problem of guessing the current player count isn't that difficult. You could only use the peak_players column, as the current player count is almost always proportional to the peak player count.